

Exceptional service in the national interest



Achieving Many-core Performance Portability with Kokkos

Christian Trott

Carter Edwards, Mark Hoemmen, Eric Phipps

Unclassified, Unlimited release



Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

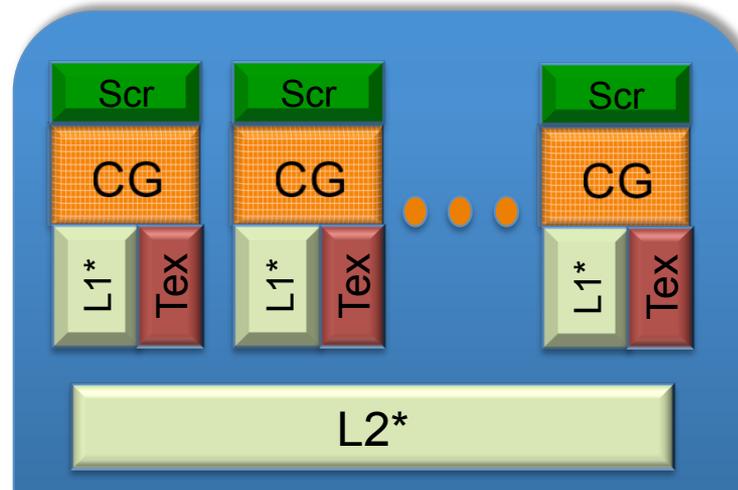
The Vision

4 Memory Spaces

- Bulk non-volatile (Flash?)
- Standard DDR (DDR4)
- Fast memory (HBM/HMC)
- (Segmented) scratch-pad on die

3 Execution Spaces

- Throughput cores (GPU)
- Latency optimized cores (CPU)
- Processing in memory

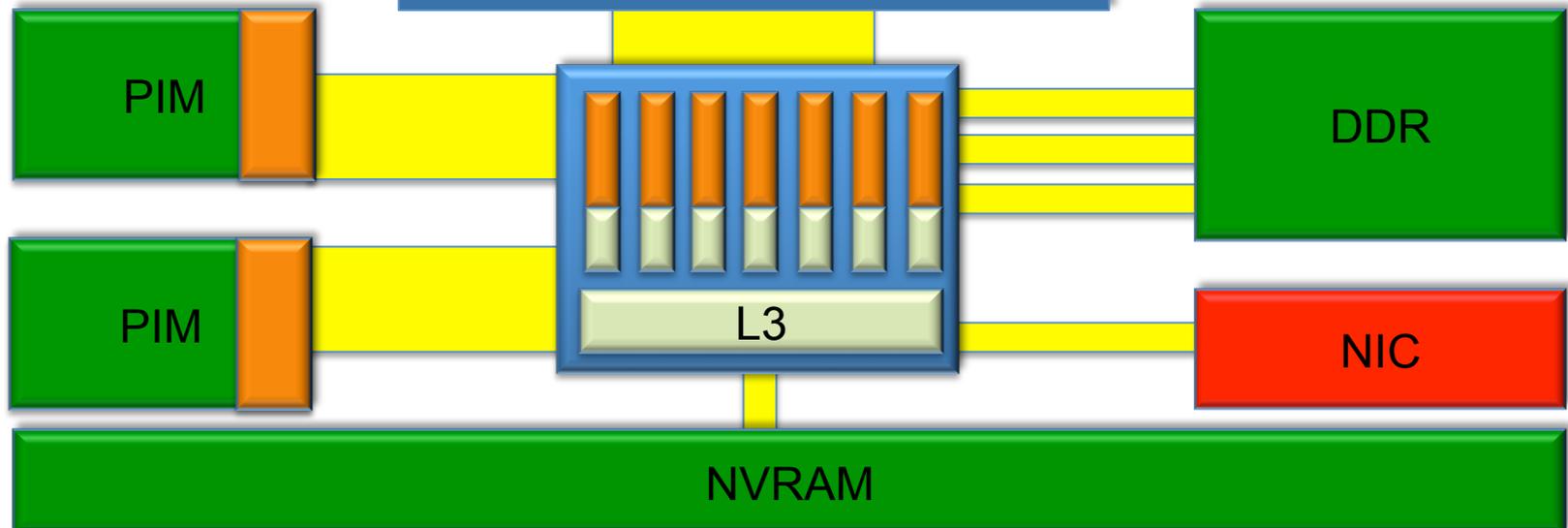


Special Hardware

- Non caching loads
- Read only cache
- Atomics

3 Programming models??

- GPU: CUDAish
- CPU: OpenMP
- PIM: ??



Kokkos

C++ template library, C++98, part of Trilinos but no internal dependencies

Data abstraction layer

- Multi-dimensional arrays
- Compile time data layouts
- Memory placement attribute
- Memory access traits
- NUMA aware
- View Semantics

Parallel dispatch

- Functor based
- *parallel_for*, *parallel_reduce*, *parallel_scan*
- Different backend models
 - Pthreads*, *OpenMP*, *Cuda*, more to come
- 2D Threading parallelism (thread-teams)
- Tasking layer under development

Memory Spaces

A 4D Kokkos View, 2 runtime+2 compile time dimensions on a NVIDIA GPU:

```
typedef View<double**[4][8], Cuda> view_type;  
view_type a("A",n,m);
```

Create a view on host with same data layout, padding etc.:

```
view_type::HostMirror b = create_host_mirror_view(a);
```

Exchange data:

```
deep_copy(a,b);
```

Control data layout:

```
View<double**[8], LayoutRight, Cuda> c;  
View<double**, LayoutTileLeft<4,4>, OpenMP> c;
```

Execution Spaces

Device concept: *Execution Space + Memory Space*

Execution Space is tied to a Backend Programming Model

e.g. Cuda + CudaSpace, Cuda + CudaUVM, OpenMP + HostSpace

A functor specifies the execution space

```
template<class view_type>
struct Dot {
    //Dispatch work to closest execution space
    typedef view_type::device_type device_type;
    view_type a,b;

    dot(view_type a, view_type b): a(a_in), b(b_in) {};

    KOKKOS_INLINE_FUNCTION
    void operator (int i, volatile double& sum) {sum+= a(i)*b(i);}

    KOKKOS_INLINE_FUNCTION
    void join(volatile double& sum, double& add) {sum+=add;}
}
```

Special Hardware

Access Traits in 4th template parameter

How is it used: e.g. Random Access, Stream, Non Caching

Shallow copy conversion: *think pointer casting*

```
view<const double**[8], LayoutRight, Cuda, Random> d = c;
```

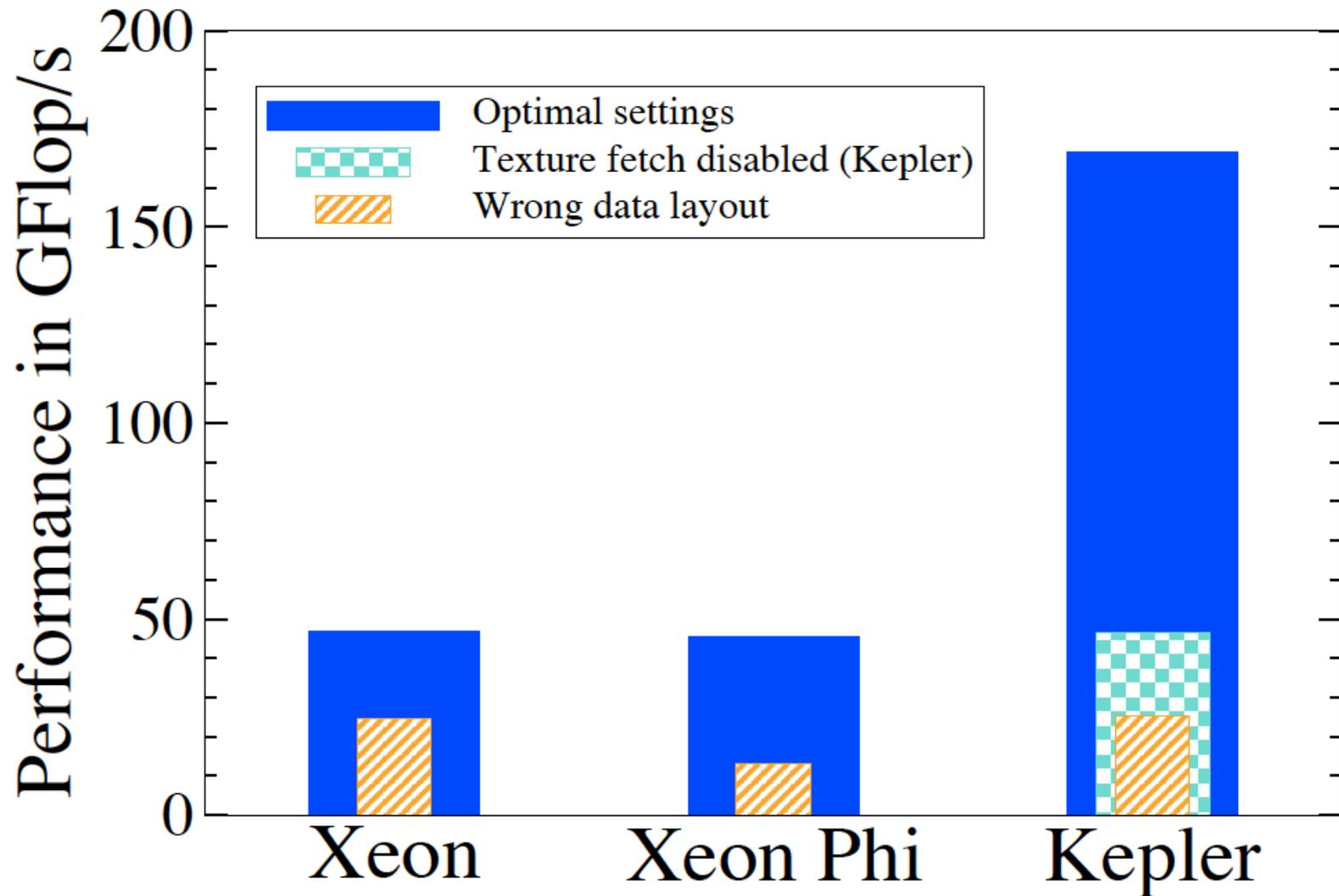
Implementation via Template specialization of access operator:

```
template< typename iType >
KOKKOS_INLINE_FUNCTION
double operator[]( const iType & i ) const
{
    #if defined( __CUDA_ARCH__ ) && ( 300 <= __CUDA_ARCH__ )
        AliasType v = tex1Dfetch<AliasType>( obj , i );
        return *(reinterpret_cast<ValueType*> (&v));
    #else
        return ptr[ i ];
    #endif
}
```

*Actual: Multiple layers with **enable_if** syntax to filter out incompatible types and deal with MemorySpace and Layout index calculation*

Impact of Layout

MiniMD Lennard Jones Force Kernel, 864k atoms, 2.5σ



(Sparse) LinAlg

Vector

- Dot, Norm, Add, Scale

MultiVector capabilities

- Dot, Norm, Add, Scale

CrsMatrix

- MatVec

TPL support

- can call MKL, CuSparse if Layout and data types allow

Some Guidelines:

- use general entry functions
- template expansion to get desired attributes (e.g. access traits)
- silent fallback for TPLs

Containers

vector

- std::vector drop-in replacement
- supports most common functions

DualView

- Holds views to two memory spaces
- Tracking/Synchronization

Unordered Map

- thread-scalable insert and retrieve

MiniFE – CG-Solve

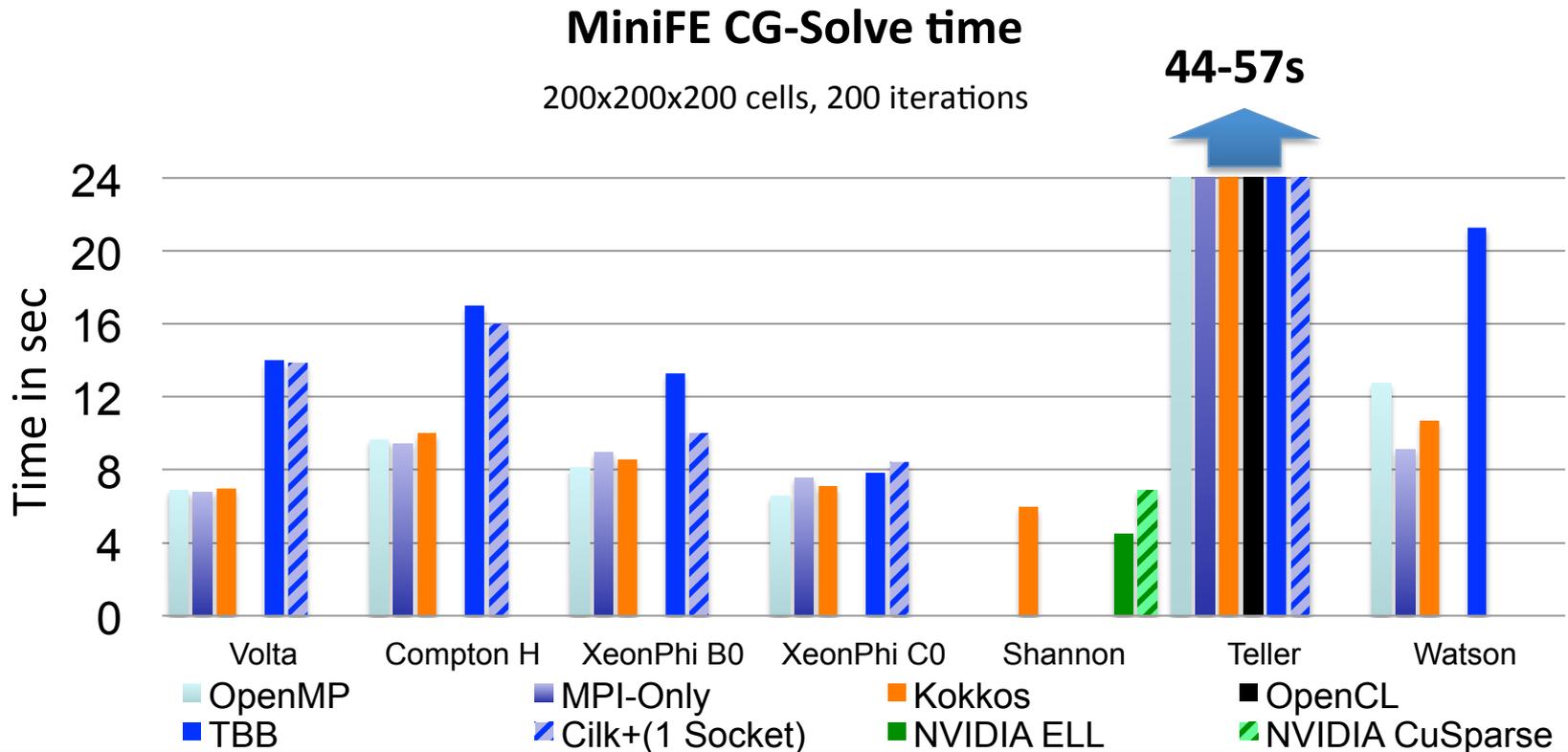
Finite element code miniApp in Mantevo (mantevo.org)

Heat conduction, Matrix assembly, CG solve

Most variants of any miniApp in Mantevo

more than 20 implementations in Mantevo repository; 8 in Mantevo 2.0 release

Models aspects of Sandia's mechanical engineering codes

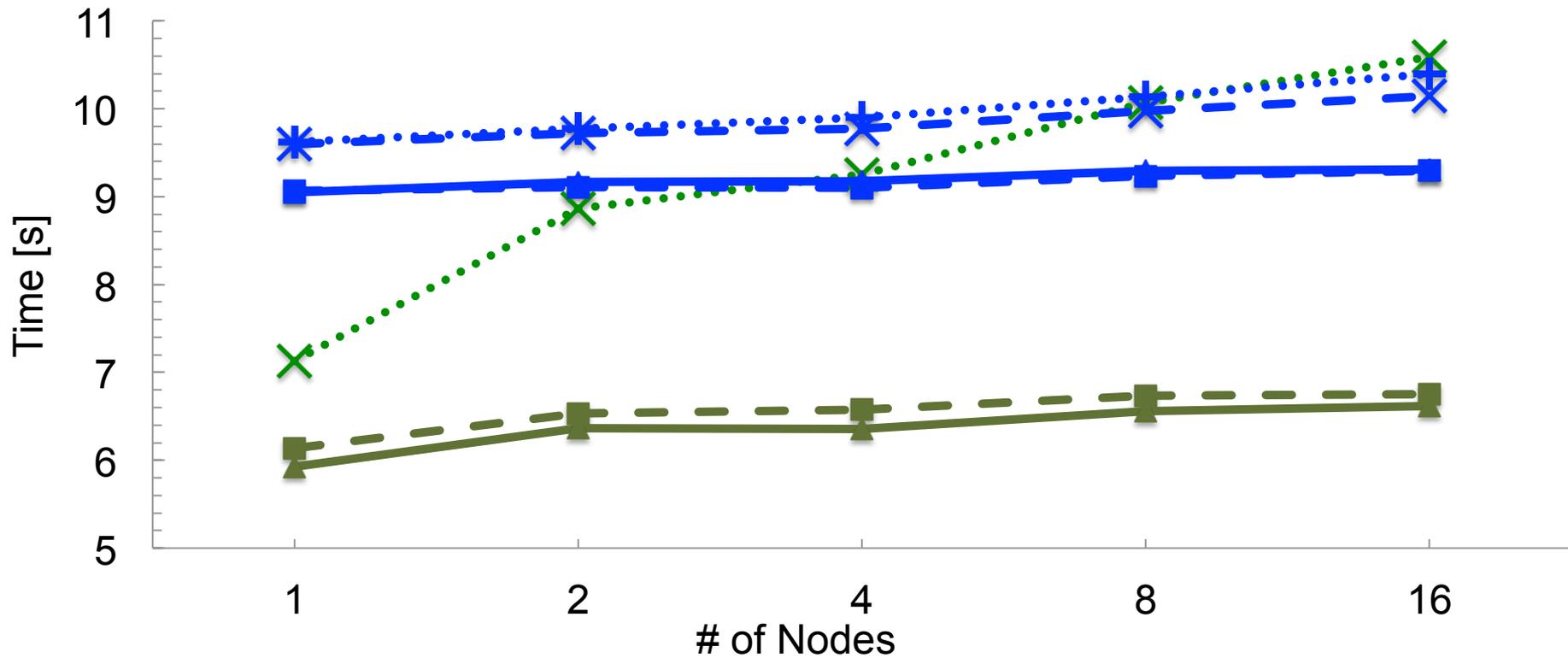


Tpetra

- Templated distributed memory layer in Trilinos
- Prior node-level parallelism by ClassicKokkos:
 - limited use(-fulness) outside Tpetra
 - no data-layouts or access traits
- **Incremental** Porting to Kokkos
 - Change Data structures
 - Port Kernels
 - Keep Backward compatibility**
- MultiVector, Vector, CrsMatrix can use new Kokkos Devices
- Basic linear algebra kernels working
- OpenMP, Pthreads and CUDA backend working
- CUDA: using UVM in CUDA 6.0 for data management
 - => reduced development time
 - => no need to identify every possible execution path with Host access

Tpetra/MiniFE – CG-Solve

200³ weak scaled, MiniFE – problem, Dual Intel SB/K20x



•×• Tpetra Cuda

•+• Tpetra Pthread

—×— Tpetra TPI

—▲— MiniFE-Cuda

—■— MiniFE-CuSparse

—▲— MiniFE-Pthreads

—■— MiniFE-MKL

Kokkos Projects / More Info

www.trilinos.org

- Trilinos with Tpetra, Kokkos etc.
- Documentation, Tutorials, Downloads

www.mantevo.org

- Mantevo Suite of miniApps (R&D 100 in 2013)
- Kokkos variants of miniMD and miniFE, more to come

lammmps.sandia.gov

- LAMMPS molecular dynamics code
- Kokkos integration started

Siam PP: MS27 Portable Manycore Sparse Linear System Assembly Algorithms
MS70 Unified Task-Data-Vector Parallelism on Manycore Architectures
MS73 Layered DSLs for Portable Manycore Performance

crtrott@sandia.gov , hcedwar@sandia.gov



**Sandia
National
Laboratories**

Exceptional service in the national interest

Questions and further discussion: crtrott@sandia.gov

The MatVec

```

template<class RangeVector, class CrsMatrix, class DomainVector, class CoeffVector1,
        class CoeffVector2, int doalpha, int dobeta >
struct MV_MultiplySingleFunctor {
    typedef typename CrsMatrix::device_type          device_type ;
    typedef typename CrsMatrix::ordinal_type         size_type ;
    typedef typename CrsMatrix::non_const_value_type value_type ;
    typedef typename Kokkos::View<value_type*, typename CrsMatrix::device_type> range_values;

    typedef MV_MultiplyShflThreadsPerRow< device_type , value_type > ShflThreadsPerRow ;

    CoeffVector1 beta; CoeffVector2 alpha; CrsMatrix m_A ;
    DomainVector m_x ; RangeVector m_y ; size_type n;

    KOKKOS_INLINE_FUNCTION
    void operator()(const size_type i) const {
        const size_type iRow = i/ShflThreadsPerRow::device_value; //What is my Row
        const int lane = i%ShflThreadsPerRow::device_value; //What is my VectorLane
        const SparseRowViewConst<CrsMatrix> row = m_A.rowConst(iRow); //get a Row
        const size_type row_length = row.length ;
        value_type sum = 0;

        if (doalpha != -1) {
#ifdef KOKKOS_HAVE_PRAGMA_IVDEP //Not all compilers support all pragmas
#pragma ivdep
#endif
#ifdef KOKKOS_HAVE_PRAGMA_UNROLL
#pragma unroll
#endif
#ifdef KOKKOS_HAVE_PRAGMA_LOOPCOUNT //Mean trick: vectorize on CPU, not on MIC
#pragma loop count (15)
#endif
            for (size_type iEntry = lane; iEntry < row_length; iEntry += ShflThreadsPerRow::device_value) {
                sum += row.value(iEntry) * m_x(row.colidx(iEntry));
            }
        }
    }
}

```

The MatVec -- continued

```
} else {  
  
#ifdef KOKKOS_HAVE_PRAGMA_IVDEP  
#pragma ivdep  
#endif  
#ifdef KOKKOS_HAVE_PRAGMA_UNROLL  
#pragma unroll  
#endif  
#ifdef KOKKOS_HAVE_PRAGMA_LOOPCOUNT  
#pragma loop count (15)  
#endif  
    for (size_type iEntry = lane; iEntry < row_length; iEntry += ShflThreadsPerRow::device_value) {  
        sum -= row.value(iEntry) * m_x(row.colidx(iEntry));  
    }  
    if (ShflThreadsPerRow::device_value > 1)  
        sum += shfl_down(sum, 1, ShflThreadsPerRow::device_value);  
    if (ShflThreadsPerRow::device_value > 2)  
        sum += shfl_down(sum, 2, ShflThreadsPerRow::device_value);  
    if (ShflThreadsPerRow::device_value > 4)  
        sum += shfl_down(sum, 4, ShflThreadsPerRow::device_value);  
    if (ShflThreadsPerRow::device_value > 8)  
        sum += shfl_down(sum, 8, ShflThreadsPerRow::device_value);  
    if (ShflThreadsPerRow::device_value > 16)  
        sum += shfl_down(sum, 16, ShflThreadsPerRow::device_value);  
  
    if (lane == 0) {  
        if (doalpha * doalpha != 1) sum *= alpha(0);  
        if (dobeta == 0) m_y(iRow) = sum;  
        else if (dobeta == 1) m_y(iRow) += sum;  
        else if (dobeta == -1) m_y(iRow) = -m_y(iRow) + sum;  
        else m_y(iRow) = beta(0) * m_y(iRow) + sum;  
    }  
}  
};
```